1. Nomenclature pour code en python

- **UTILISER UN DICTIONNAIRE param POUR STOCKER LES PARAMETRES DU ROBOT/SIMULATION**
- q : angle articulation
- dq : vitesse articulation
- q_d : Position désirée ou de référence
- tau : couple moteur
- tau_control : couple de commande à appliquer
- e : erreur entre q et q_ref
- e_der : erreur dérivé
- e_int : erreur intégré
- nb_joint: nombre d'articulation
- nb_samples: nombre d'échantillon d'une trajectoire
- q_m_all : vecteur avec tous les éléments de q "mesurés" par CoppeliaSim stockés de taille (nb_joint x nb_samples)
- e_all : vecteur avec tous les éléments d'erreur stockés
- t : temps de fonctionnement
- ts : temps d'échantillonnage

2. Installation Coppeliasim

https://www.coppeliarobotics.com/ version **EDU**

Tout a été testé sur la version V4.5.1 (https://www.coppeliarobotics.com/previousVersions)

Pour utiliser avec python :

1. Avoir le dossier zmqRemoteApi dans le dossier des codes python
2. Ouvrir ce dossier dans vscode
3. pip install cbor
4. pip install zmq

Pour la version V4.9.0 le dossier zmqRemoteApi n'est pas necessaire. Cela peut s'installer via pip de la forme : pip install coppeliasim-zmqremoteapi-client

Par contre dans l'initialisation (foncionts à la suite de ce paragraphe, il faut remplacer

```
from zmqRemoteApi import RemoteAPIClient
```

par

```
from coppeliasim_zmqremoteapi_client import RemoteAPIClient
```

Plus d'informations dans :
https://manual.coppeliarobotics.com/en/zmqRemoteApiOverview.htm

3. Fonctions utilisées pour CoppeliaSim

- Initialisation :

```
from zmqRemoteApi import RemoteAPIClient
client = RemoteAPIClient()
sim = client.getObject('sim')
client.setStepping(True)
```

- Récupérer le handle d'une articulation :

```
q_handle = sim.getObject("/name_joint")
```

- Initier la simulation :

```
sim.startSimulation()
```

- Lire la position d'une articulation avec son handle :

```
q=sim.getJointPosition(q_handle)
```

- Ecrire la position d'une articulation avec son handle :

```
sim.setJointPosition(q_handle,ang) #avec ang en radians
```

- Attacher un objet à l'effecteur en utilisant leur handles :

```
sim.setObjectParent(object_handle,end_effector_handle,'true')
```

- Libérer l'objet :

```
sim.setObjectParent(object_handle,-1,'true')
```

- Récupérer la matrice homogène qui exprime la pose de l'effecteur dans le repère monde :

```
T_ee=sim.getObjectMatrix(end_effector_handle,sim.handle_world)
```

- Appliquer la matrice pose (par rapport au monde) à un target de Coppelia monde :

```
sim.setObjectMatrix(target_handle,base_handle,pose)
```

- Initialiser l'environnement pour calculer la cinématique inverse (à placer juste après les handles des éléments coppeliasim) :

```
sim_ik = client.getObject('simIK')
```

```
ik_env=sim_ik.createEnvironment()
ik_group=sim_ik.createIkGroup(ik_env)
sim_ik.setIkGroupCalculation(ik_env,ik_group,sim_ik.method_damped_least
_squares,1e-1,100000)
sim_ik.addIkElementFromScene(ik_env,ik_group,sim.handle_world,tcp_handl
e,target_handle,sim_ik.constraint_position)
```

- Appliquer la cinématique inverse à la scène (à placer dans la boucle ou au moment d'implémenter les modifications de la scène)

```
sim_ik.applyIkEnvironmentToScene(ik_env,ik_group)
```

- Avancer un pas dans la simulation (à inclure dans la boucle while)

```
client.step()
```

- Lire les forces et couples d'un capteur d'effort placé dans la scène

```
result_f, force_m, torque_m = sim.readForceSensor(force_sensor_handle)
```

(Attention, result_f est simplement un binaire pour confirmer la mesure, les autres deux paramètres de sortie sont les vecteurs de 3 composantes avec forces et couples respectivement)

- Arrêter la simulation :

```
sim.stopSimulation()
```

4. Classe et fonctions utilisées pour les moteurs Myactuator

Important : Ne pas utiliser joint torque sans consulter à l'enseignant !

- Initialisation de la communication
  - Sur le terminal écrire :

```
sudo ip link set can0 up type can bitrate 1000000
```

    Il faut le faire à chaque fois qu'on branche l'USB-CAN

- Importer la classe moteur :

```
from resources.utils import RobotController
```

- Créer un objet de la classe moteur :

```
ROBOT = "PENDULUM" #define type of robot PENDULUM/3DDL/SCARA
robot = RobotController(ROBOT, USE_TOOL)
```

Les types de robot sont dans resources/robot_config/robotConfig ainsi que des paramètres comme son ID, Km, limites articulaires et de vitesse

- Définir la position initiale comme zero du robot

```
while True:
        a = input("mise à 0 ? (y): ")
        if a.lower() == 'y':
            break

    robot.init_offset() # To put the robot at zero in the initial position
```

- Finir le programme :

```
    # End of your code
    time.sleep(2)
    robot.shutdown()
```

- Lire l'état du robot :

```
robot.get_joint_position() # (in deg)
robot.get_joint_velocity() # (in deg/s)
robot.get_joint_torque()# (in N.m)
```

ATTENTION, les valeurs que son récupères se mettent à jour quand on commande le robot. Si pas de commande, pas de mis à jour !

- Controller le robot :

```
robot.set_joint_position() # for position control (in deg)
robot.set_joint_velocity() # for velocity control (in deg/s)
robot.set_joint_torque() # for torque control (in N.m)
```

ATTENTION, vous ne devez pas faire les trois, juste 1 en fonction du type de commande à utiliser